Parallel Discrete Event Simulation Techniques for Scientific Simulations

A Thesis
Presented to
The Academic Faculty

By

Jagrut Dave

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Computer Science

Georgia Institute of Technology

May, 2005

Parallel Discrete Event Simulation Techniques for Scientific Simulations

Approved by:

Dr. Richard Fujimoto, Advisor
College of Computing
Georgia Institute of Technology

Dr. Santosh Pande
College of Computing
Georgia Institute of Technology

Dr. Kalyan Perumalla
College of Computing
Georgia Institute of Technology

Date Approved: April 18, 2005

ACKNOWLEDGMENT

I am greatly indebted to Dr. Richard Fujimoto for his guidance and support throughout my master's study. It was under his mentoring that I developed a focus and became interested in parallel and distributed discrete event simulations.

Special thanks go to Dr Kalyan Perumalla. As my co-advisor, he helped and encouraged me all along the way. I also thank Dr. Santosh Pande for his insightful comments and suggestions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Exponential growth in computer technology, both in terms of individual CPUs and parallel technologies over the past decades has triggered rapid progress in large scale simulations. However, despite these achievements it has become clear that many conventional state-of-the-art techniques are ill-equipped to tackle problems that inherently involve multiple scales in configuration space. One difficulty is that conventional ("time driven" or "time stepped") techniques update all parts of simulation space (fields, particles) synchronously, i.e. at time intervals assumed to be the same throughout the global computation domain or at best varying on a sub-domain basis (in adaptive mesh refinement algorithms).

Using a serial electrostatic model, it was recently shown that discrete event techniques can lead to more than two orders of magnitude speedup compared to the time-stepped approach. In this research, the focus is on the extension of this technique to parallel architectures, using parallel discrete event simulation. Previous research in parallel discrete event simulations of scientific phenomena has been limited

This thesis outlines a technique for converting a time-stepped simulation in the scientific domain into an equivalent parallel discrete event model. As a candidate simulation, an electromagnetic hybrid plasma simulation is considered. The experiments and analysis show the trade-offs on performance by varying the following factors: the simulation's model characteristics (e.g. lookahead), application's load balancing, and accuracy of simulation results. The experiments are performed on a high performance cluster computer, using a conservative synchronization mechanism. Initial performance

results are encouraging, demonstrating very good parallel speedup for large-scale model configurations containing tens of thousands of cells. Overheads for inter-processor communication remain a challenge for smaller computations.

1. INTRODUCTION

This chapter provides an overview of computer simulation and its applications, the motivation and objectives of this research, a summary of results and the organization of the rest of the thesis document.

1.1 Simulation

Simulating a system involves reproducing the behavior of the system over time. In particular, we are concerned with software simulations where the state of the system is stored in computer memory, and computation is used to change this state as time advances. At the end of a simulation, one can generate statistics of the system without having to physically construct it.

Simulation has long been a widely used approach for analyzing systems, especially when it comes to systems where mathematical models are too complex to be solved analytically. For example, consider a scientist who wishes to design a more efficient simulation of the earth's magnetosphere. In order to evaluate different aspects of the magnetosphere, simulation can be used to answer questions concerning the electric and magnetic fields surrounding the earth where classical mathematical modeling approaches are intractable. Simulations can include necessary information about the magnetosphere such as ion density, temperature, pressure, solar radiation, etc. For a given initial condition, the simulation can give precise information about the state of the magnetosphere over time. This information can be used to identify regions with high

electric and magnetic fields that might be encountered by satellites and astronauts stationed in space.

There are two major paradigms for simulation. In a *discrete event simulation* state changes in such systems are modeled by events that occur at discrete points in time. In the previous example, events might correspond to movement of an ion, beginning of solar flare, etc. By contrast, in *continuous (time-stepped) simulations* state changes are viewed as occurring continuously over time. For example, simulations used for weather forecasting typically use continuous simulation methods. The behavior of such systems is usually described by a set of differential equations. This research will focus on the use of discrete event simulation techniques as an alternative to time-stepped simulation in simulating physical systems.

1.2 Motivation

Historically, the fields of continuous and discrete event simulation have been largely distinct, with limited cross-disciplinary interaction. Simulations mixing continuous and discrete models, e.g., to model circuits and hybrid simulations that mix discrete and continuous models are notable exceptions [1]. Time-stepped simulations have traditionally been used in continuous simulations to model physical systems described by partial differential equations. On the other hand, event-stepped simulations have their origins in operations research and management science, and have since have found application in war gaming and telecommunications.

A key question is whether Discrete Event Simulation (DES) can be used to model systems such as plasmas that have a large number of states. Codes developed as part of

recent research confirm that this is not only possible, but that DES plasma codes can be much faster than their existing time-stepped counterparts. DES has the advantage that it relaxes processing through the use of irregularly time-stamped events that only update *what* needs to be updated *when* it needs to be updated. Further, DES decouples complex models by allowing separate portions to evolve independently over simulated time. In this way it offers the potential for diverse model behaviors to be represented in a more natural manner than time-stepped simulation.

With the increasing availability of off-the-shelf cluster hardware, there is a great amount of interest in parallel execution to speed up complex, time-consuming simulations. Parallel DES (PDES) exploits concurrent event processing and has been shown to be a promising method for this purpose [2]. Parallel discrete event simulations must be comparable to their time-stepped counterparts in terms of accuracy of results, but are expected to be more efficient. Techniques are needed to enable domain experts to exploit the benefits of parallel discrete event simulation without having to deal with its implementation issues. The separation of domain modeling issues from issues associated with parallel discrete event simulation would facilitate the distributed development of parallel simulation code. A majority of simulations in the scientific domain are based on the time-stepped methodology for state evolution. Techniques to generate a parallel discrete event simulation from time-stepped models can have a far-reaching impact in the scientific domain.

1.3 Research Objectives

While exploitation of DES techniques can provide dramatic performance improvements, by itself, DES is not sufficient to achieve the performance and scalability objectives necessary for large scale multi-physics simulations. A principal objective of this research is to realize parallel execution by exploiting PDES techniques on a large-scale parallel cluster of computers. A goal of this research is to illustrate a process for converting complex time-stepped scientific models into efficient and accurate parallel discrete event models. To evaluate the feasibility of this process, the modeling of a hybrid plasma simulation is examined as a case study.

1.4 Summary of Results

The research described here is concerned with the realization of efficient parallel discrete event simulations of complex plasma physics models. The specific contributions of this research are in the development and performance evaluation of the new model codes. Previous research had focused on generating high-fidelity time-stepped or discrete event models of hybrid plasma [43-49]. Issues arising due to the parallel and distributed execution of discrete event plasma models are unique to this research. A summary of results is as follows:

- *DES Modeling*. An electromagnetic hybrid plasma model is considered that is a complex time-stepped model based on partial differential equations. The data structures and events for the corresponding discrete event code are presented. The work presented here highlights issues associated with the DES modeling of plasmas, but may be applicable to other scientific models.

- *Distributed Implementation*. A distributed implementation of the hybrid plasma model has been developed using discrete events and a conservative synchronization mechanism.
- *Performance Evaluation.* The hybrid code was thoroughly tested over a cluster of 128 computers connected by Gigabit Ethernet. Properties such as scalability, load balancing and parallel simulation overheads were investigated.
- *Performance Optimization*. The initial implementation used a simplistic mapping from model entities to simulation entities. It had large computational and synchronization overheads and generated a large number of messages. Hence, the model did not scale very well. The cause of this inefficiency was identified and a different mapping technique was developed that reduced both computational and message passing overheads.

1.5 Organization of Thesis

The remainder of the thesis is organized as follows. Chapter 2 summarizes related work, including PDES methodologies and novel relaxations to synchronization mechanisms that provide improved performance at the cost of reduced accuracy in the simulation results. Chapter 3 describes in detail a case study of an electromagnetic hybrid plasma simulation model and the mechanism for event-based time advances used in this model. Chapter 4 describes the performance evaluation of the hybrid plasma model over a cluster of computers. Chapter 5 summarizes the conclusions and discusses areas of future research.

2. RELATED WORK

PDES systems have been the subject of intensive research for the past two to three decades. This chapter presents an overview of important PDES concepts and terminology. Simulation techniques for PDES systems that attempt to improve parallel performance at the cost of accuracy are discussed. Next, the N-body simulation problem is described that is used to study properties of complex physical systems and has been an area of active research for many decades.

2.1 Parallel and Distributed Discrete Event Simulation

One can view a physical system as consisting of interacting physical processes. Each physical process has state that is modified when certain actions occur. When the state changes, new actions either for it or for other physical processes may occur. These actions will happen at specific points in simulation time.

A DES can simulate such a physical system by assigning a Logical Process (LP) to model each physical process [2]. The state of an LP is the set of variables that represent the state of the physical process. Each LP includes software that models the effect of the corresponding actions in the physical system. Events are represented in the simulation by exchanging messages between LPs. Events occur at distinct points in simulation time, and thus, messages are time-stamped accordingly. Ideally, in order to properly model the physical system, the simulation must process all events in time-stamp order. Otherwise, undesirable *causality errors* may occur where future events affect those in the past. *Synchronization* of events is needed to ensure such causality errors do not occur.

Parallel and distributed discrete event simulation refers to the execution of a discrete event simulation program over multiple processors. *Parallel* discrete event simulations execute on tightly coupled computer systems such as shared memory multiprocessors. All processors are interconnected via high-speed switches, and hence, communication latencies are low. On the other hand, *distributed* discrete event simulations are executed on loosely coupled computer platforms, such as workstations interconnected via commercial local or wide area networks (LANs and WANs). Communication latencies are substantially higher than those for tightly coupled systems, and are usually at least an order of magnitude higher on distributed computing platforms. Depending on their use, parallel and distributed discrete event simulations can be classified into two categories:

- *Analytic Simulations.* Simulations used to analyze systems must mimic the causal relationships (i.e. before and after relationships) of a physical system precisely or as closely as possible. They are typically used to obtain accurate statistics concerning the behavior of the physical system being modeled. Hence, one of the key characteristics of analytical simulations is repeatability, i.e., the simulation should always yield the same outputs if the same input parameters and initial LPs' states are used. Also, these simulations typically run in an as-fast-as-possible manner, in order to complete the simulation run as quickly as possible. Finally, analytic simulations often run without any user interactions, except for the fact that a user is allowed to observe the outputs and state of the modeled physical system during the simulation execution. An example of such a simulation is modeling a telecommunication network. Simulation is often used in an iterative manner to evaluate and/or verify a network design. She/he may be interested in

understanding how the network performs under different conditions, and revise the network design to maximize performance or reliability.

- *Distributed Virtual Environment Simulations.* This is a second category of simulation applications. These simulations are used to create a virtual world, e.g., for training or entertainment applications, that appear sufficiently realistic to its participants, to meet the objective of the exercise. Before and after relationship may not always be perceptible by human participant, so casual relationships can sometimes be relaxed. Typically, these simulations run in real-time since humans are "immersed" in the virtual world, e.g., interacting with entities in the model. The required degree of realism depends on the purpose of the simulation.

Analytic and distributed virtual environment simulations often require different levels of accuracy, as was seen from the above examples. This leads to different requirements with respect to the ordering of events in a parallel or distributed discrete event simulation. Consequently, different synchronization mechanisms are typically used for these categories of simulations. Synchronization algorithms for analytic simulations will be discussed later.

2.2 Time Management

The execution of a distributed simulation involves at least three distinct notions of time:

1. *Physical Time.* This refers to time in the physical system, i.e., the actual system being modeled. Recall that physical system can be viewed as a collection of interacting physical processes, where the state of the system evolves over physical time.

2. *Simulation Time.* This is the representation of the physical time in the simulation. We use simulation time to assign time-stamps to events during the execution of the simulation. In a discrete event simulation state changes occur at discrete points in simulation time.
3. *Wall clock Time.* This refers to the time during which the simulation is executing on a computer platform. It is the time obtained by reading the computer's real-time clock.

In simulations executing in as fast as possible mode there is usually no direct relationship between wall clock time and simulation time. On the other hand, advances in wall clock and simulation time are typically paced in synchrony in distributed virtual environment simulations.

Processing events from all LPs in time-stamp order guarantees that no causality errors occur. This is easily accomplished in a sequential simulation. One need only maintain a list of all unprocessed events in the system, and process them in non-decreasing time-stamp order. The simulator removes the event with the smallest time stamp from the list, and processes that event. Processing an event includes performing a computation that models the behavior of the corresponding physical process when that event occurs. Thus, processing an event typically results in changing the state of the LP. As a result, one or more new events may be generated for other LPs. These new events are inserted into the event list.

In a parallel or distributed simulation the execution of the LPs may be distributed across different CPUs. At first glance, the original simulation paradigm where LPs exchange messages fits this mode of execution very well. Events are processed in time-

stamp order in each LP, and events generated for other LPs are sent to CPUs containing those LPs. If all events, i.e. both local events and those received from other LPs, are processed at each LP in non-decreasing time stamp order, then this mode of execution yields the same result as an equivalent sequential simulation.

2.2.1 The Synchronization Problem

Some mechanism is required to ensure that each LP processes events in non-decreasing time-stamp order. Without such a mechanism, nothing prevents a situation where an LP processes an event from the event list, and later receives an event with a time-stamp smaller than the one it has already processed. Events need to be synchronized to ensure this does not happen.

But what events need to be synchronized and what events can be processed concurrently? If two events affect the same state, they must be synchronized. For example, two events on the same LP that modify the same portion of the LP's state cannot be executed concurrently. Events on different LPs may also have to be synchronized, since processing an event at an LP may generate an event for another LP, and hence indirectly affect the state of another LP. Generally, it is difficult to know ahead of time the events that will be generated during the simulation execution and which LPs' states they will affect. However, it suffices for each LP to process events in time-stamp order, in order to guarantee the same results are produced as in the sequential execution.

Clearly, the partitioning scheme of the physical system into physical processes that are mapped to logical processes determines how much concurrent execution can be achieved. This is an important task in modeling the system, but will not be discussed here. Rather, we concentrate on synchronization of the system once the LPs have been

defined and mapped to processors. Generally, there are two approaches to synchronization termed *conservative* and *optimistic* discrete event simulation.

A conservative simulation ensures no event will be processed by an LP until it can guarantee that no event with a smaller time stamp will be received by that LP. Because an event could cause a message to be sent to every other LP with the same time stamp, this could lead to very poor performance. To overcome this problem, the concept of *lookahead* is introduced. One approach is to assign a lookahead value to each LP. When an event is being processed, all new events generated as the result of processing that event must be at least that LP's lookahead value in the future. For example, processing an event with time-stamp T at an LP with lookahead L can only result in new events with a time-stamp greater than or equal to T + L.

How does lookahead help in the concurrent processing of events? Consider a situation where all LPs have the same lookahead. If T is the smallest time-stamped event in the entire system, then we can safely process events with time stamps less than T + L and guarantee no LP processes events out of time stamp order.

Optimistic simulation is an alternative approach to synchronization. While lookahead is a straightforward concept for concurrent processing, it sometimes imposes difficult constraints on the model. In addition, the lookahead may be too small to achieve acceptable levels of parallelism for some systems. For these reasons, many have examined optimistic synchronization methods where an event may be processed despite the fact that later an event with smaller time-stamp may arrive. To ensure the simulation's final state is the same as that obtained in a sequential execution, state saving and rollbacks are introduced. When an LP receives an event in its past, it must roll back

to a previously saved state. In addition, if there were any events or messages generated for other LPs by rolled back events, they also must be cancelled. This is accomplished using an anti-message mechanism [2]. When an LP receives an anti-message, it will also have to rollback to one of its previously saved states, and generate anti-messages if the cancelled event has already been processed.

One issue in optimistic simulation is to know how optimistic LPs can execute, or in other words, how far LPs can advance ahead of each others during the execution. It may happen that advancing too far ahead results in too many rollbacks, resulting in much wasted computation. Secondly, there is a need for efficient state saving and rollback techniques.

A detailed overview and discussion of both conservative and optimistic techniques can be found in [2]. The Null Message algorithm by Chandy and Misra [6] is one of the first conservative approaches. Null messages are used to provide other LPs with a Lower Bound on Time Stamp (LBTS) of future messages sent from one LP to another. Too many null messages may degrade the performance of such systems, and later approaches such as [7-8] address this issue. In [9], the LBTS is computed by taking into account the time of the next unprocessed event and its lookahead, allowing LPs to advance simulation time faster. Some recent conservative techniques are described in [10-15]. The Cai-Turner carrier-null scheme [10] tries to resolve the problem of transmitting redundant null messages due to low lookahead cycles in digital circuit simulations. The Deblocking Event Algorithm [11] uses a mathematical model of the structure of the simulation network to reduce overheads. The algorithm presented in [12] selects the next event to be processed through co-operative interaction among a group of LPs. The shared resource

algorithm [13] attempts to avoid deadlocks and artificial blocking while the critical channels approach [14] improves on event scheduling, cache behavior and load balancing. A framework for automatically computing lookahead for conservative simulations is presented in [15]. In [16], the authors measure the cost of conservative synchronization and [17] presents techniques for load balancing.

The initial optimistic synchronization approach, based on rollbacks and anti-messages, was the Time Warp algorithm developed by Jefferson [18]. A constrained optimistic scheduling technique that allows out of order processing based on a Moving Time Window protocol is presented in [19]. The authors use transitive dependency tracking in [20] and conditional knowledge in [21] for efficient optimistic simulations. Performance evaluation of Time Warp and its variants is presented in [22-23] and [24] discusses the cost of copy state saving and rollback mechanisms. In [25], the authors use an adaptive throttling mechanism to reduce the overheads associated with roll backs in Time Warp simulators. Unified frameworks that enable conservative and optimistic execution through a single interface are presented in [26-28].

2.3 Distributed Simulation Architecture

As shown in Figure 1, a distributed simulation system can be divided into a layered architecture consisting of two components: (1) the application models and (2) the parallel simulation executive that manages events and the progression of simulation time. The rationale for this architecture is that heterogeneous simulations could be *federated* using a middleware that provides a well-defined interface [28]. Also, in the case when model developers are domain experts and simulation executive developers are computer

scientists, such a partitioning allows for concurrent development of different PDES components.

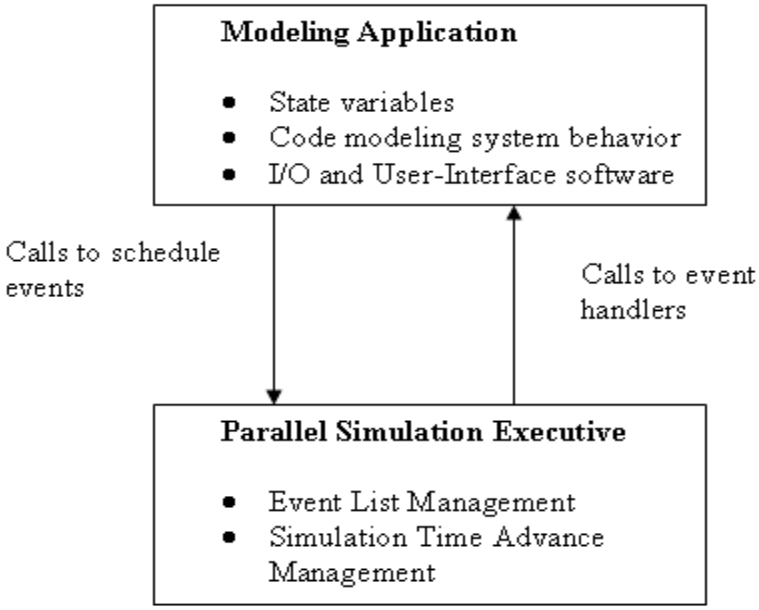

Figure 1 Components of a PDES System

2.3.1 Modeling Application

As a case study in this research, we develop and use electromagnetic hybrid plasma models. The state of the LPs in the model consists of values of electric and magnetic field vectors. Multiple LPs execute concurrently and communicate with each other via the parallel simulation executive through event exchanges.

2.3.2 Parallel Simulation Executive

The parallel simulation executive provides support for event delivery and synchronization (time management) which are necessary for PDES. It serves as middleware for

supporting distributed simulation applications, and ensures repeatable event processing across the entire simulation. Each event is assigned a time-stamp. Events are delivered to simulation applications according to their time-stamps. Events processed by an application may be generated locally by the application or sent by remote applications. A synchronization mechanism is required to ensure global event ordering. Synchronization techniques for PDES systems have been discussed in previous sections. The operation to recover a previous state in an optimistic parallel simulation is known as a *rollback,* and involves undoing incorrect computations. A widely used technique for supporting rollback is *state-saving* that saves the values of state variables prior to an event computation and restores them from saved values upon rollback. *Copy state saving* creates an entire copy of a process's state, while *incremental state saving* keeps a log of changes to individual state variable. A relatively new technique for rollbacks, *reverse computation* [29], realizes rollbacks by performing the inverses of the individual operations executed in the event computation. These techniques have been exploited in small and large-scale parallel simulations.

2.4 Distributed Simulation of Scientific Models: Challenges

Development of next generation scientific codes requires innovations in application modeling as well as the parallel executive components. The hybrid plasma model used as a case study presents specific modeling challenges that must be addressed. First, the computation is highly dynamic. Dependencies between events, a critical issue for parallel processing, are difficult to predict with great certainty. Although much of the simulation

is highly localized, in some regions, effects can propagate at speeds as high as the speed of light, resulting in low lookahead. For plasma simulations, lookahead also depends on initial values of other model parameters. Static assumptions about lookahead can have catastrophic effects when unforeseen events are generated. Designing the simulation application to maximize its lookahead often leads to complex and unmanageable code.

Existing PDES techniques either require sufficient lookahead or optimistic synchronization to achieve efficient parallel execution. The applications presented here represent a challenging test case for existing parallel simulation software. For example, in a recent feasibility study using the Synchronous Parallel Environment for Emulation and Discrete-Event Simulation (SPEEDES) system [27], the experiments that were conducted ran into memory and performance limitations, with SPEEDES requiring over an order of magnitude more memory to store particle state information, as compared to a time-stepped version of the code [44]. The amount of parallelism changes dramatically during the execution, e.g., the number of pending events may be small in the early phases of execution, but very large in later phases. The computation must be able to adapt as the execution proceeds. The parallel simulator must be able to efficiently manage data structures containing large numbers (millions) of events, and minimize the amount of storage that is required. The results must be in tune with those obtained by using the corresponding time-stepped version of the code. Finally, another problem to be addressed is scalability over a large number of processors.

Most work to date in PDES is based on assigning precise time stamps to events, and time stamp order event processing. Time intervals have been studied as a means for specifying temporal or other uncertainty in simulation applications [30-31]. Fuzzy logic

has also been used [32]. A new primitive for process-oriented simulations called Interval Hold was proposed in [33]. Pre-sampling [34] allowed conservative time stamp ordered event processing and enhanced lookahead by drawing random numbers from a distribution. A combination of optimistic and conservative techniques called the Local Time Warp mechanism was proposed in [35]. It hierarchically combined a conservative Time Window technique with the Time Warp algorithm. Unsynchronized simulation [36] has been studied to determine the impact of violating causality on different types of simulations. The next section presents Approximate Time Stamps, a popular technique for relaxed synchronization that uses time intervals. Preemptive Event Processing, a relatively new technique developed in the context of plasma simulations, is presented later.

2.4.1 Approximate Time Stamps

The approach presented in [37] allows modelers to use time intervals rather than precise time stamps to specify uncertainty as to when events occur. Partial orderings called Approximate Time (AT) and Approximate Time Causal (ATC) order are proposed and synchronization algorithms developed that exploit these specifications to yield more efficient execution on parallel and distributed computers. Performance measurements of the AT-ordering mechanism on a cluster of workstations demonstrate as much as twenty-fold performance improvement compared to time stamp ordering with negligible impact on the results computed by the simulation. The context for much of this work is federated simulation systems that provided the initial motivation for this work. These results demonstrate that exploiting temporal uncertainty inherent in the simulation model can lead to efficient parallel execution despite zero lookahead using conservative

synchronization techniques, a long-standing problem in the PDES field. Application of this research to scientific models would deal with the zero-lookahead problem in scientific models as well as the performance requirements.

2.4.2 Preemptive Event Processing

In plasma physics simulations, the simulation domain is divided into smaller portions (simulation units) called cells. Each cell is modeled as a simulation application that sends and receives events. Cell-based physical events typically reschedule themselves during the execution. Unlike other discrete event simulations (e.g. air traffic control, military simulations) plasma physics simulations can adaptively alter their sequence of events without compromising the correctness of results, provided the causality and accuracy constraints in the simulation time continuum are maintained. This property plays a significant role in determining synchronization strategies for concurrent execution of PDES systems of plasma physics models.

In conventional DES systems, a simulation application advances its local simulation time according to the sequence of events that it processes. The Preemptive Event Processing (PEP) algorithm introduces a finite time window extending into the future from the global Lower Bound on Time-Stamp (LBTS) value [53]. LBTS is the smallest time-stamp value among the time-stamps of all events in the PDES. An application is allowed to process all events falling within this window starting at the LBTS value, without updating its local time for each event processed. The time window adapts itself, depending on the time-stamp values. An interesting observation is that if the PEP time window is small compared to the characteristic time delays with which events reschedule themselves, appreciable parallel speedup can be obtained. PEP execution enables time-

constrained PDES execution that reproduces physically correct results while providing significant savings in execution time compared to conventional time-driven parallel codes. Determining the optimum PEP window width at run-time is an on-going research.

2.5 N-Body Simulation Problem

The classical N-body problem simulates the evolution of a system of *N* bodies, where the force exerted on each body arises due to its interaction with all the other bodies in the system. The electromagnetic hybrid model, considered as a case study in this research, consists of ions interacting with other ions and the medium that contains them. These interactions are based on well-known properties of physical bodies and are expressed as partial differential equations. They are similar to interactions in an N-body simulation. The N-body simulation proceeds over time steps, each time computing the net force on each body and thereby updating its position and other attributes. If all pair-wise forces are computed directly, this requires $O(N^2)$ operations at each time step. The basic approach is a simulation: loop forever, stepping discretely through time and do the following at each time-step:

1. Update positions using velocities $(x_{i+1} = x_i + \Delta t\, v_i)$
2. Calculate forces F
3. Update velocities $(v_{i+1} = v_i + (1/m)\, \Delta t\, F_i)$

The focus is usually on various methods used to calculate the forces. It is possible to use multiple resolutions for time, i.e., different Δt for different particles. Hierarchical tree-based methods have been developed to reduce the complexity. There are many approaches to solving the N-Body simulation, some of which are the following [38]:

1. *Particle-Particle (PP):* The simplest and naive method. Accumulate forces by finding the force *F(i,j)* of particle *j* on particle *i,* integrate the equations of motion (including accumulated forces), update the time counter and repeat for the next time step. Although this is straight-forward, a constant time-step in the integration could lead to overflow errors - this can be avoided with a numerical integration scheme that uses variable time-steps that cuts down the time-step when the particles are near each other and increase the time-step when they are far away from each other. Computationally it is also expensive: $O(N^2)$ operations are required to evaluate the forces on all *N* particles.
2. *Mesh Based Methods:*
    - *Particle-Mesh (PM):* This method treats the force as a *field quantity* by approximating it on a mesh. The approach is to break space into a uniform mesh and place each particle in the appropriate cell of the mesh. Uses a discrete Fourier transform to solve a partial differential equation over the mesh. For a mesh of *m* cells, this costs *O (m* log *m)* time.
    - *Particle-Particle Particle-Mesh (P3M):* A combination of *PP* for close particles and *PM* for distant particles. Provides better accuracy for roughly the same time as *PM.*
3. *Tree Based Methods:*
    - *Top-down particle-cell (Appel, Barnes-Hut):* Divides space into an *octtree*. When a cell is far enough away from a particle we can calculate the force to the center of mass of the cell instead of having to calculate the

force to each particle in the cell. It costs *O (N* log *N)* operations for *N* particles.

- *Bottom-up particle-cell (Press):* Similar to top-down except that the *octtree* is created bottom-up by organizing particles into *nearest-neighbor* groupings. It costs *O (N* log *N)* operations for *N* particles.

4. *Cell-cell (Fast Multipole Method, FMM):* This also uses a tree to model particles, but it allows for *"cell-cell"* interactions as well as *"particle-cell"* interactions. These are the fastest known methods for solving N-Body problems and cost *O(N)* time for *N* particles in a uniform distribution.
5. *Individual Time-Step Scheme (ITS):* In this method, each particle has its own time-step Dt_i and maintains its own time t_i. To integrate the system, we first select the particle for which the next time ($t_i + Dt_i$) is the minimum. Then, its position is predicted at this new time. The same is done for all other particles at this same time ($t_i + Dt_i$). Then the force on that particle from other particles is calculated using the gravitational law formula.

Gravitational N-body simulation can be used to model many astrophysical systems. However, although there are many applications of Gravitational N-body simulation, it does not necessarily mean that a single algorithm can be used to study various systems. The GRAPE special-purpose system [39] was specifically designed to run the *ITS* N-body simulation algorithm at a very high speed using specialized hardware.

3. CASE STUDY

This chapter describes in detail the candidate distributed simulation application, a simulation of a hybrid plasma physics model. The modeling techniques that were used to convert the time-stepped application model into a discrete event model are presented. Modeling state updates of the application as a sequence of events and distributing the simulation over multiple processors requires the services of a parallel simulation executive. The mapping of application level entities to simulation processes (or LPs), the application's data structures and simulation events are described in detail. The parallel simulation middleware used in this research was Musik [27]. A related DES modeling technique for plasma codes was presented in [43], which considered the spacecraft charging problem.

3.1 The Courant-Friedrichs-Levy Condition

In most explicit numerical simulations, physical quantities are advanced in time based on their values at previous instances in time. Therefore, the simulation time step cannot cause a state update at a larger time interval than the limit imposed by the most rapidly changing process in the model. For propagation problems, this limiting factor is the well-known Courant-Friedrichs-Levy (CFL) condition. Physically this condition means that information cannot propagate in a discrete mesh faster than one cell in one simulation cycle (time step) without causing numerical instability. As a result, in such systems all cells and/or particles can only be advanced with a time step meeting the most restrictive condition existing in the system because it is not known *a priori* whether each actual

computation carries significant information. In contrast, the DES model introduced in [43] provides a temporal mesh for every computational entity, enabling asynchronous time integration of the system state variables. At any given time a DES model enables its entities to evolve independently through simulation time. Time steps for idle entities may effectively become infinite. The state of the simulation at any given time is the union of the states of individual entities. In essence, conventional codes solve time-driven equations,

$$df / dt = RHS(f, t)$$

while the DES approach focuses on solving their event-driven analogs,

$$dt / df = RHS^{-1}(f, t)$$

Discrete Event Systems Specification [2] is a related framework that provides a way of simulating systems based on differential equations.

3.2 The Hybrid Model

Modeling physical system phenomena often requires time integration of continuous systems described by partial differential equations (PDEs) or first principles (e.g. particle behavior). PDE-based computational models typically operate in a discretized configuration space (mesh) and solve finite difference or finite element equations defined for each mesh cell. In addition, Particle-In-Cell (PIC) simulations represent certain system components by macro particles, i.e., Lagrangian elements with the ability to move in configuration space. In computational plasma physics, macro particles form large clusters of charged particles (e.g. protons, electrons etc). In fluid dynamics, particles correspond to blobs with distinct macro-characteristics. Modern large-scale PIC

simulations typically employ millions of mesh cells, with about 30-100 particles per cell [45].

The candidate hybrid model considered here is an electromagnetic hybrid algorithm, with fluid electrons and kinetic ions. The charge and mass of an ion are both normalized to one. Electrons are considered to be a homogeneous fluid, with ions contained in it. Electrons do not have a mass, but exert an electric and magnetic field on the ions. The model is ideally suited for physical phenomena that occur on ion time and spatial scales. Maxwell's equations are solved by neglecting the displacement current in Ampere's law (Darwin approximation), and by explicitly assuming charge neutrality. In addition to the algorithm considered here, there are several variations of electromagnetic hybrid algorithms with fluid electrons and kinetic ions [48-50]. The model considered here uses the one-dimensional resistive formulation [42] that casts field equations in terms of vector potential. The model problem uses the piston method where incoming plasma moving with flow speed larger than its thermal speed is reflected off the piston located on the rightmost boundary. This leads to the generation of a shockwave that propagates in the direction opposite to the one in which the plasma was initially flowing. A flow speed is used that is large enough to form a fast magneto-sonic shock. The plasma is injected with a normalized velocity of 1.0. The background magnetic field is tilted at an angle of 30^{o}. This version of the code does not strictly conserve flux. However, the lack of strict local flux conservation does not change the result significantly in the problem of interest.

The simulation domain is divided into cells [43], and the ions are uniformly loaded into each cell. Ions move from left to the right in the domain, and new ions are inserted into the leftmost cell to keep the flow constant. For the purposes of this feasibility study,

only a one-dimensional model is considered. It is a spatial grid and field equations are finite differenced as in standard time-stepped simulations. However, discrete event methodology is used for time advancement of the simulation. Figure 2 shows the domain.

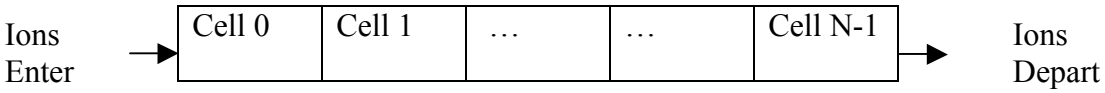

Figure 2 Simulation Domain

The hybrid model extends the PIC model by introducing background fields that affect particle motion. As in conventional (time-driven) simulations, the parallelization of event-driven continuous PIC models is realized by decomposing the global computation domain into sub-domains. In each sub-domain, individual cells and particles are aggregated into containers that may be mapped to different processors. The parallel execution of time-driven simulations is commonly achieved by copying field information from the inner lattice cells to ghost cells of neighboring sub-domains and exchanging out-of-bounds particles between processors at the end of each update cycle. By contrast, in parallel *asynchronous* PIC simulations particle and field events are not synchronized by each time step. They do not take place at the same time interval throughout the simulation domain, but occur at arbitrary time intervals, introducing synchronization challenges. Unless precautions are taken, a process may receive an event message from a neighbor with a simulation time stamp that is in its past.

3.3 The Parallel Simulation Executive

The parallel simulation executive used in this work is called Musik [27]. Musik is a parallel simulation engine that handles synchronization and communication among the applications. It reduces the burden of the application developer by not requiring an understanding of underlying PDES synchronization mechanisms. More information about Musik can be found in [27]. The parallel simulation is composed of a collection of Simulation Processes (SPs) that communicate by exchanging time-stamped event messages. An SP is the equivalent of a logical simulation process (or an LP). Each cell is mapped to an SP. The state of an SP includes electric and magnetic field vectors for its cell. An asynchronous global reduction-based conservative synchronization algorithm is used for ensuring time-stamp order event delivery [51]. Conservative synchronization ensures that a simulation process never receives an event in its past [2]. However, runtime performance is critically dependent on apriori determination of the application's lookahead. The lookahead is roughly dependent on the degree to which an LP can process its events and advance its local simulation time without having to wait for other LPs. In this simulation, the lookahead is specified before execution begins and remains constant throughout the lifetime of the execution. The lookahead value is the same for all LPs. The determination of the lookahead value is described in section 3.5.2. In the sections that follow, the current simulation time of an LP or a cell is referred to as *current time* and the constant value of the hybrid model's lookahead is referred to as *lookahead*.

3.3.1 Load Balancing

Load balancing is an important issue for scientific models. As with any parallel or distributed application, the computation must be evenly balanced across processors and inter-processor communication should be minimized to achieve the best performance. Often these are conflicting goals. This is particularly challenging in PDES because of the irregular, unpredictable nature of state updates. The "load" in this application is the number and type of events. Each event type requires a different amount of computation and occurs with a different frequency. Load balancing also affects the efficiency of synchronization mechanisms. For example, poor load distribution can lead to overloading some LPs which would cause other LPs to block and wait for them to complete their processing. The load-balancing algorithm used in this research is the Region-Deal algorithm, as described next.

The Region-Deal load balancing scheme is implemented as follows. The simulation domain is sequentially divided into equal-sized groups of cells. Processors are allocated a range of contiguous cells, called a "Region", from each group. A processor gets at least one Region from a group and each processor is allocated the same number of cells (or SPs). This ensures that portions of the domain that are more active and represent greater computational load are uniformly mixed with those that have less load. The Region-Deal mapping of SPs to processors is carried out before the simulation begins and remains fixed throughout the lifetime of its execution.

3.4 Model Data Structures

Model data structures consist of those used by an SP for managing event scheduling and state updates. An SP can schedule an event on another SP at a simulation time greater than or equal to its *current time + lookahead.* In this case, SPs are cells and events include state updates, ion transfers and notifications of field change. Events are explained in detail in the sections that follow.

There are two priority queues associated with each SP – the IonQ and PendQ (see Figure 3). Both store ions that are contained within the cell that is associated with the SP. Each is sorted by MoveTime, the time at which the ion is expected to move to an adjacent cell. The ions in IonQ are moving relatively slowly and will not move to a new cell until a later time in the simulation. The PendQ consists of two types of ions:

1. Ions that have moved out from the cell. These are kept temporarily and removed when the cell advances its simulation time.
2. Ions that have been scheduled to move within the cell or move out from the cell, i.e., events for moving those ions have been scheduled.

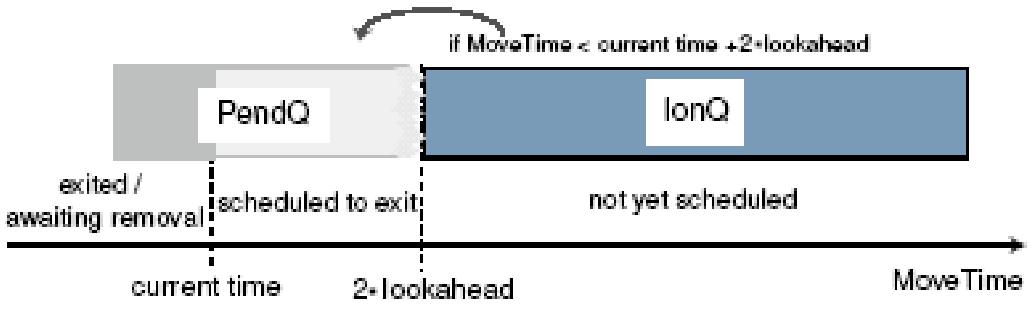

Figure 3 Priority Queues of an SP

Initially, each cell of the domain contains a fixed number of ions. The default value is 100 ions per cell. These ions are inserted in the cell's IonQ. The movement of an ion is modeled as an event. A cell needs to schedule an event to move an ion at least one lookahead period before the ion's MoveTime. Ions with MoveTimes greater than or equal to *current time* + *lookahead* are scheduled to move. They are removed from the IonQ and inserted into the PendQ. The upper bound on the value of MoveTime for scheduling the movement of an ion is twice the lookahead interval. Thus, for any given value of *current time*, ions with MoveTime in [*current time* + *lookahead, current time* + *2* lookahead*] are scheduled to move. This ensures that an ion is scheduled at least one lookahead period, but no more than two lookahead periods before its MoveTime.

The two priority queues can be viewed as a continuous queue of ions sorted by their MoveTime, as shown in Figure 3. Having two different queues for ions simplifies event scheduling. When the field values of a cell change by more than a predefined threshold, MoveTimes of ions in its IonQ are re-calculated. Ions that have already been scheduled to move based on previous field values are kept in the PendQ and are not affected by this change. Since field updates are also modeled as events, this allows the model to deal with cyclic causality of event scheduling, i.e., field update events causing ion motion events and vice versa. In conventional DES systems, event retractions are used to retract previously scheduled events. Considering the large volume of events that the hybrid model generates, event retractions would prove costly in terms of computation and extra messages. Having two separate queues for ions allows the model to maintain causal relationships between events without using retractions. The field values of a cell include its electric and magnetic field vectors. A cell also maintains a copy of the field values of

its immediate left and right neighbors. It uses these copies along with its old values and considers ions from both the IonQ and PendQ in calculating new field values.

3.5 Simulation Events

The simulation advances the state of each cell via three main types of events. A cell is the basic unit of simulation. Events are generated by cells and scheduled on cells. They are handled asynchronously by their destination cells. Following an object-oriented design, events and ions are modeled as objects. Events can have data, which represent the information to be conveyed from the source cell to the destination cell. The interaction of the three event types with Musik is illustrated in Figure 4.

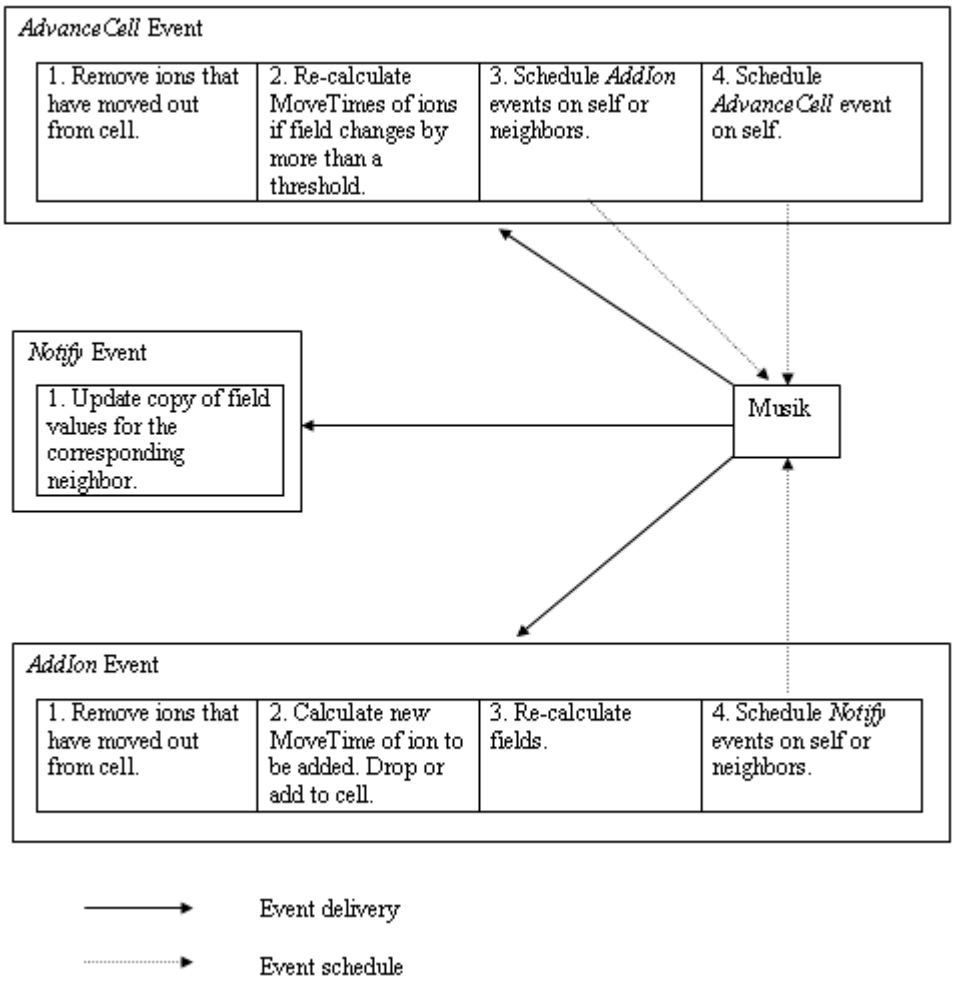

Figure 4 Simulation Events

The three event types are described below:

AdvanceCell Event:

The *AdvanceCell* event is scheduled by a cell only on itself. Initially, every cell schedules this event to begin the simulation. This event is responsible for updating field values of the cell and advancing the cell's simulation time. It does the following:

1. Those ions that have been sent to neighboring cells are removed from the PendQ. This is done by removing all ions from the PendQ that have MoveTimes less than

the *current time* of the cell. For the leftmost cell, new ions are inserted into its IonQ in order to keep the flux of incoming ions fixed at the left boundary. Field values are re-calculated after adding each ion.

2. If the magnetic field has changed by more than a predefined threshold, the MoveTimes of ions in the IonQ are updated. If the MoveTime of an ion becomes less than *current time* + *lookahead*, it is not possible to schedule an ion transfer event for that ion without violating the lookahead constraint. In this case, the ion's MoveTime is changed to *current time* + *lookahead.* The IonQ is sorted such that the ion with the minimum MoveTime is on top.
3. Ions from the IonQ with MoveTimes less than *current time* + *2*lookahead* are scheduled to move. This is done by scheduling an *AddIon* event (described later) for each one of these ions. These ions are removed from the IonQ and inserted into the PendQ.
4. An *AdvanceCell* event is scheduled by the cell on itself for a simulation time equal to the minimum MoveTime of ions in the IonQ minus the *lookahead*. This ensures that the next *AdvanceCell* event for that cell occurs *lookahead* time before the earliest MoveTime of ions in the IonQ. This would allow the ion with the earliest MoveTime to be moved (as in step 3 above) when the new *AdvanceCell* is processed.

AddIon Event:

When an ion is scheduled to move, an *AddIon* event is scheduled by the source cell on the destination cell. The destination cell could be the source cell, if the ion is stepping through the cell or it could be one of its immediate neighbors. This event is used to model

ion transfers across cells or ion motion within a cell. The argument to this event is the ion to be moved. The following actions occur in the destination cell:

1. Ions that have moved out of the cell are removed from the PendQ. This is done by removing all ions from the PendQ that have MoveTimes less than the *current time* of the cell.
2. The next MoveTime of the ion to be added is calculated. If its MoveTime is less than the minimum MoveTime of all ions in the IonQ of the cell, the ion is dropped. This happens when the source cell schedules an *AddIon* event based on a copy of the destination cell's field values, but before the event is delivered to the destination, the destination's field values change by more than a predefined threshold. This causes a re-calculation of MoveTimes of ions in the destination cell's IonQ. If the ion is not dropped, its *AddIon* event at its MoveTime would violate the lookahead constraint. The *current time* of the cell is less than the minimum MoveTime of all ions in the IonQ by *lookahead*. An event to move an ion can only be scheduled for a simulation time greater than or equal to the cell's *current time* + *lookahead.* If the ion needs to be moved before *current time* + *lookahead,* the *AddIon* event for this ion would have to be scheduled for a time less than *current time* + *lookahead,* which is not possible. If the ion to be added has a new MoveTime greater than the MoveTime of the top of the cell's IonQ, it is added to the cell's IonQ.
3. Fields are calculated again. The new ion's charge is included in this calculation if it has not been dropped.

Notify Event:

This event is scheduled by a cell on its immediate left and right neighbors when its field values are re-calculated. It is indirectly triggered by *AdvanceCell* and *AddIon* events. Its data consists of updated electric and magnetic field vectors of the source cell. It is scheduled at *lookahead* time into the future. Each cell maintains a copy of the field values of its left and right neighbors. When a cell receives a *Notify* event, it replaces its copy for the corresponding neighbor with the argument of the event.

3.6 Move Time

The MoveTime of an ion is the time at which it would exit its current cell or move within the current cell. The direction of an ion's motion is dependent on its position within the cell and values of electric and magnetic fields at that point. The electric and magnetic fields are considered to be constant within a cell, with arbitrary orientation and magnitude. An ion's model has an equation of motion that can be calculated analytically and has the general form $R(t) = At^2 + Bt + rc\ sin(ct + \_) + C$, where $R(t)$ is the position of the ion. Newton's method is used to solve for the exact exit time when the ion moves across the cell boundary.

3.7 Lookahead

The simulation application can specify a range of lookahead values, starting from zero. If the typical velocity of an ion is v, and a typical cell width is x, then the time it takes for an ion to cross a cell is x/v. The time taken by the first ion to exit a cell defines the upper bound on the lookahead value.

If lookahead is a period smaller than this time, an ion would cover a small fraction of the cell width in one lookahead period. It would have to be moved multiple times before it crosses the cell boundary. This would result in an accurate simulation of the ion's path across the cell. On the other hand, if the lookahead is too small, the parallel performance will be poor. There would be fewer event computations during a lookahead period and synchronization overhead, especially in a distributed execution, would reduce the speedup. Thus, selecting a lookahead value is a tradeoff between accuracy and speedup and it can be used to "tune" the simulation appropriately.

4. EXPERIMENTAL PERFORMANCE EVALUATION

This chapter presents the experimental methodology and performance benchmarks for the hybrid simulation using Musik as the simulation engine. The comparison of simulation results between time-stepped and event-stepped hybrid models was presented in [43]. It was found that the event-stepped model was two orders of magnitude faster than the time-stepped model and that the difference in their results was within statistical fluctuations associated with changes in the noise level in hybrid codes. The experiments in this section are concerned with the properties of distributed execution of the hybrid code. Impact of lookahead, scalability, communication overhead and efficiency of load balancing were the key characteristics that were investigated. The results show considerable success for a one dimensional model and uncover hidden dependencies between modeling and implementation techniques for parallel and distributed plasma simulations. An optimization is proposed that enables the plasma simulation to run twice as fast as the original parallel discrete event version, without introducing any inaccuracies in the result.

4.1 Experimental Methodology

Initial parameters for the simulation included end time, lookahead, number of cells, number of ions in a cell, width of the cell, motion parameters, fluid model parameters and tolerances. These parameters were supplied as variables that were initialized with the same values that were used in the time-stepped version of the model. This would allow us

to treat the time-stepped and event-stepped models as "black boxes" whose results could be directly compared.

Additionally, the event stepped model had to be supplied a lookahead parameter. The domain is one dimensional. The minimum domain size was 8,192 cells. To enable the simulation to exhibit the kind of complexity that a 3-D domain offers, the number of cells was increased to 65,536 cells and the cell width was reduced by an order of magnitude. Table 1 lists the initial simulation parameters and their values.

Table 1 Initial Simulation Parameters

Parameter	Typical Initial Value
End simulation time	1,000s
Lookahead	0.15s
Magnetic field tolerance	7e-6 Tesla
Number of cells	65,536
Ions per cell	100
Cell width	0.25 (normalized)

For the simulation engine, a conservative global reductions based algorithm was used. Each cell in the hybrid model was mapped to an LP, so that cells would be able to communicate among themselves by event exchange.

4.2 Hardware Platforms

The hardware for scaling and load balancing experiments consisted of two clusters at the Georgia Tech High Performance Computing laboratory. The Hebrides cluster has 8 nodes, each with 4 2.8 GHz Xeon processors and 4 GB of RAM. It has 100 Mbps LAN

interconnect. The Jedi cluster is a 17 node cluster, with each node having 8 550 MHz CPUs and 4 GB of RAM. It has Gigabit Ethernet interconnect. For single processor experiments, individual nodes from the Hebrides cluster were used. For each experiment, the respective nodes were reserved to avoid context swapping overhead. One Musik federate was run per processor, as suggested by the architecture of Musik.

4.3 Impact of Lookahead

For conservative synchronization mechanisms, it is widely recognized that the concurrency achieved by the distributed simulation depends on its lookahead. Large lookahead leads to better speedup relative to a sequential DES. Lookahead is a constraint on a simulation application that requires any new event to be scheduled at least a certain amount of simulation time into the future. It is a property of the physical system and is difficult to extract in complex simulations, as it tends to be implicitly defined by source code interdependencies. Here, the effects of changing the lookahead on both the accuracy of the results as well as the execution time are evaluated. The simulation uses 4 Musik Federates (one per processor), each with 2 Regions and 512 cells per Region. The hardware for these experiments was the Hebrides cluster. Figure 5 shows variations in the spatial profile of *Btot*, with lookahead.

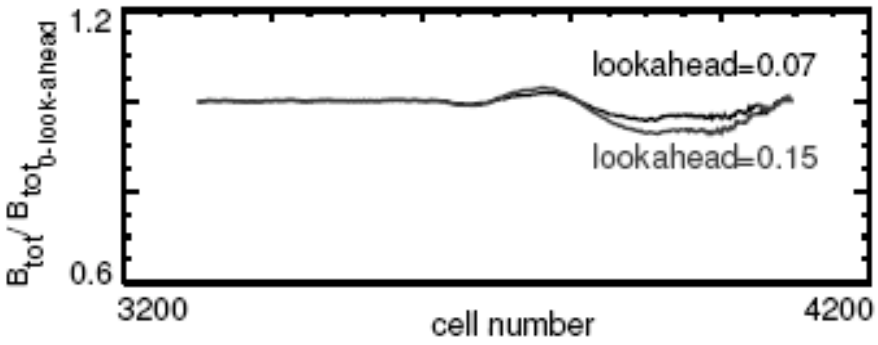

Figure 5 Impact of Lookahead on Accuracy

The zero lookahead run yields the most accurate result and is treated as a baseline. In the hybrid algorithm, the maximum lookahead must be less than the exit time of the first ion that is scheduled to exit a cell. This is approximately 0.15 for our choice of initial values of simulation parameters. Deviations of the profile from the baseline are less than 10%, even when the maximum lookahead value is used. Figure 6 shows the speedup in execution time relative to the zero lookahead run.

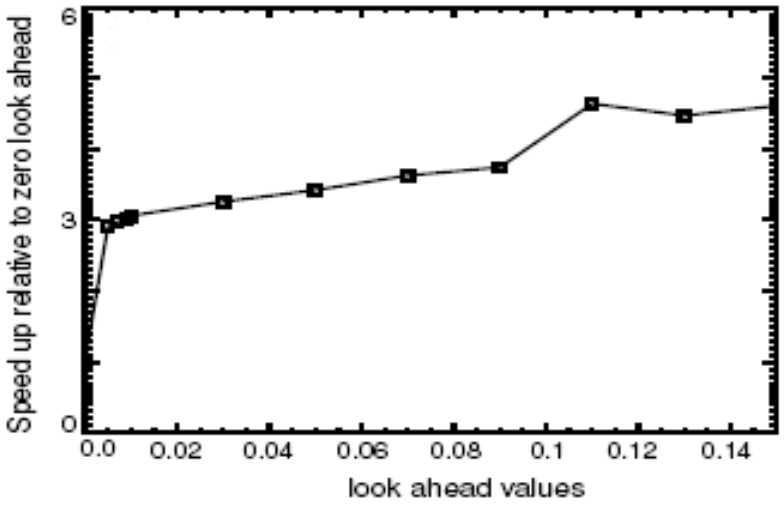

Figure 6 Impact of Lookahead on Speedup

The important point from this figure is that even small departures from zero lookahead lead to substantial improvements in speed. In fact, the most dramatic speedup (a factor of 3) is achieved when lookahead is changed from 0 to 0.005. Further changes in lookahead do improve performance, but at a much slower rate. For example, increasing the lookahead by an order of magnitude from 0.005 to 0.05 leads to only an additional 15% speedup.

4.4 Simulation Scalability

In order to deal with the complexity of multi-scale multi-physics simulations, a 3D model is desired. As a simple means to evaluating the parallel execution of a 3D model, we considered simulations with up to 65,536 cells, with 100 ions per cell. The domain size and other simulation parameters (lookahead, load balancing parameters, etc) were kept constant and the simulation was distributed over more processors. This was sufficient to identify the key issues of parallel execution. The hardware for these experiments was the Jedi cluster. Figure 7 shows the speedup as a function of the number of processors up to 128. The speedup is measured with respect to a sequential run, on a single processor.

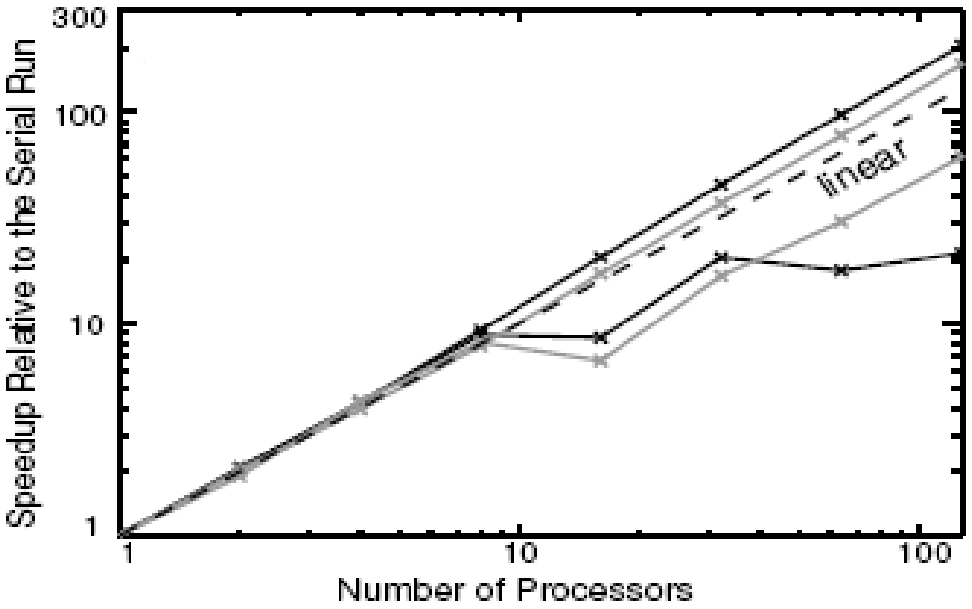

Figure 7 Scaling Performance

The dashed line is a linear scaling curve. Speedup for a domain size of 8,192 cells is shown in black and that for a domain size of 65,536 cells is in gray. For each domain size, there are two curves - speedup considering the overall execution time and speedup without considering communication time. As is evident from Figure 7, the parallel speedup is good until 8 processors, but declines as more processors are added. This is due to the architecture of the cluster, which uses a collection of 8 processor computers communicating through TCP/IP. With up to 8 processors, the entire simulation uses shared memory, and the communication overheads are low. However, with more than 8 processors, the overheads associated with TCP/IP begin to offset the speed gained by using more processors. This reduces the slope of the curve. For 8,192 cells, the speedup does not increase significantly after 32 processors. This is because increased

synchronization costs negate the gains from parallel processing. Processors do not have a sufficient computational load between global synchronizations and spend a greater fraction of time waiting on other processors. For 65,536 cells, there is enough computation between global synchronizations to obtain good speedup up to 128 processors. Since the overheads associated with inter-processor communication become relatively small as the simulation size increases, we do not anticipate this effect being as pronounced with larger, 3D simulations. In 3D simulations, each processor would have several orders of magnitude more cells, making the relative overheads associated with TCP/IP much less significant.

Communication time is obtained by subtracting the time spent in processing events from the total execution time. It includes time spent in synchronization, message queuing, blocking and all other operations that ensure correct event ordering across the simulation. Communication time and total time are measured by the simulation engine. When only the event processing time is considered, the speedup is better than expected, and is super-linear. Doubling the number of processors more than doubles the execution speed. Memory performance (specifically, cache performance) is a well-known cause of super-linear speedup. The total amount of cache memory increases in proportion with the number of processors used. By keeping the problem size constant but distributing it over more processors, the cache footprint in each processor shrinks. With enough processors, one can, for example, fit the entire computation into the processors' caches. Another extreme example is when the problem is so large that it does not fit into the memory of a single processor, causing excessive paging. In our experiments, the average time to

process an event decreases when the number of processors increases as each processor is assigned a smaller portion of the domain.

4.5 Communication Overhead

Figure 8 shows the percentage of time spent in communication and blocking for the two domain sizes mentioned above. These experiments were also conducted on the Jedi cluster.

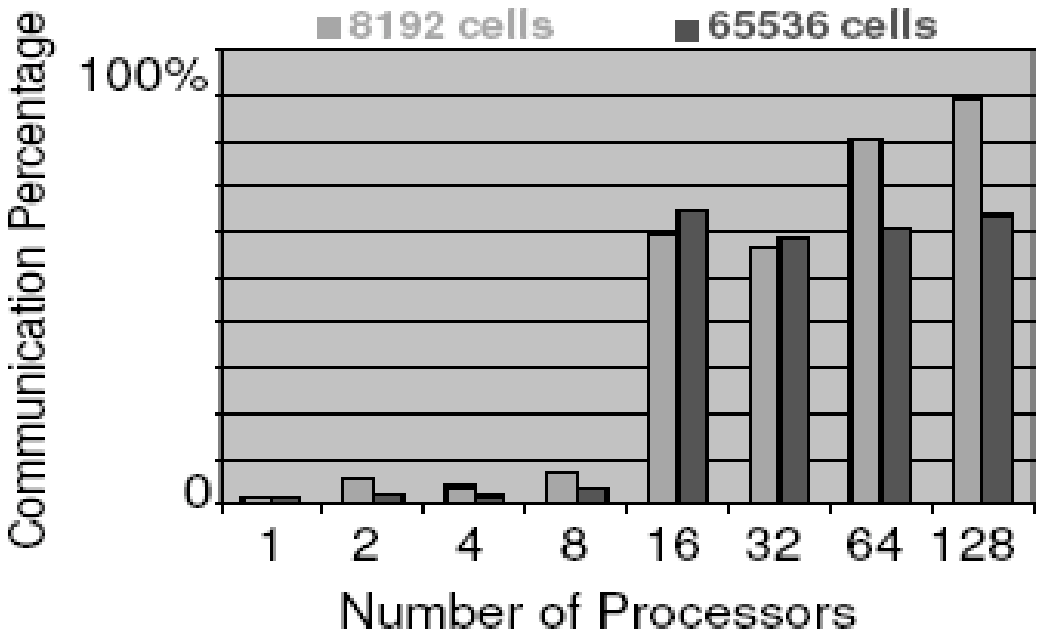

Figure 8 Communication Overhead

There is a significant increase in the fraction of execution time spent in communication and blocking for more than 8 processors. For 65,536 cells, the percentage settles to around 60% for a higher number of processors. However, for 8,192 cells, the

percentage keeps on increasing until 90% for 128 processors. The cluster architecture necessitates TCP/IP based communication for more than 8 processors, which substantially increases the delay for sending events to and synchronizing with remote processors.

The simulation engine carries out LBTS computations for synchronizing all the LPs. The LBTS algorithm finds the minimum time-stamp among the time-stamps of all the pending events in the distributed simulation. In Musik, this is implemented as a Global Reduction algorithm that uses butterfly-style communication [51]. Figure 9 shows the number of LBTS computations performed as a function of the number of processors.

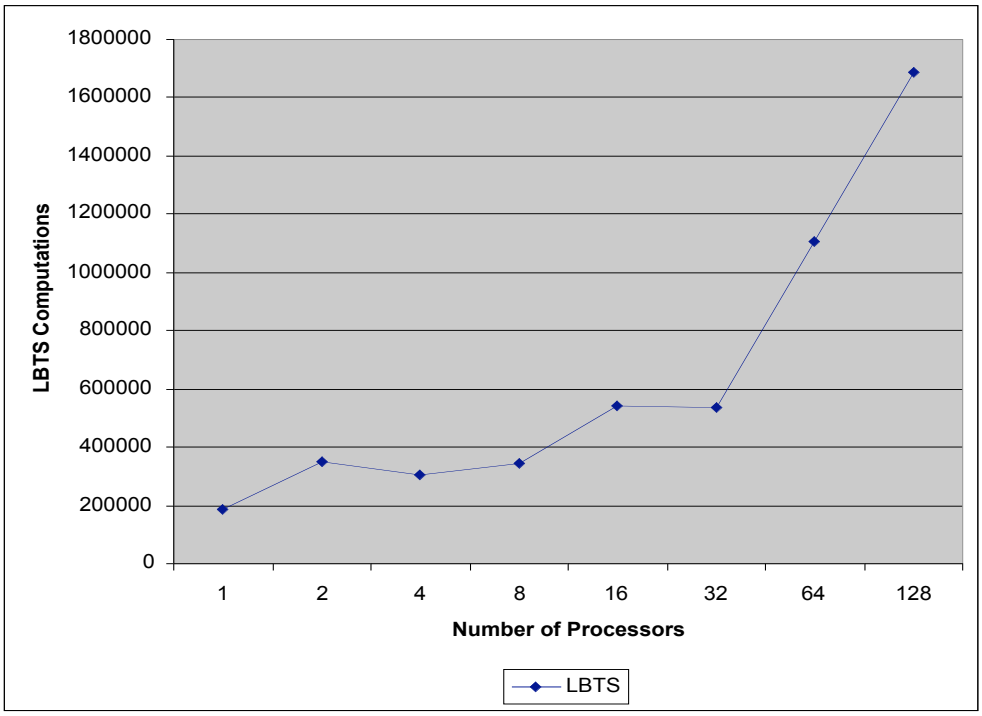

Figure 9 LBTS Computations and Scaling

The number of LBTS computations increases substantially for more than 32 processors. This, coupled with the fact that communication with remote LPs requires TCP/IP, results in the poor scaling performance of the simulation. For higher number of processors, each processor spends more time waiting on other processors to process their events before it can start processing its own events.

4.6 Load Balancing

Figure 10 shows the variation in execution time as a function of the number of Regions per processor, as distributed by the Region Deal algorithm. These simulation runs were performed on the Hebrides cluster. It was observed that all three types of events occur roughly with the same frequency during a simulation run.

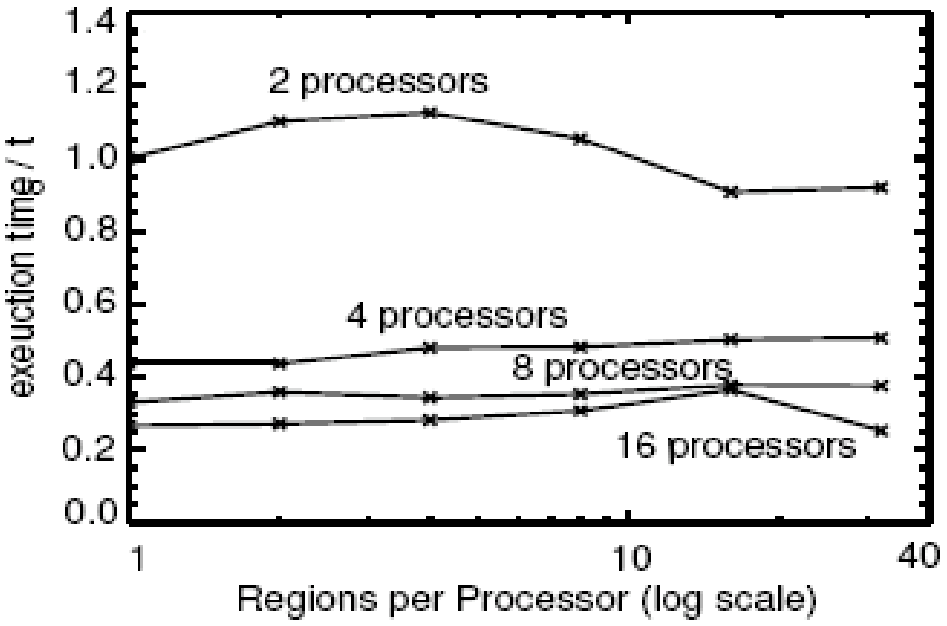

Figure 10 Performance of Load Balancing Algorithm

The curves show a different trend for a larger number of processors (4,8,16) than for the 2 processor case. For higher numbers of processors, the variation in execution time using the Region Deal load balancing scheme is less pronounced. The best execution times are close to the execution time for one Region per processor, with variations of less than 1,000 seconds. Also, increasing the number of Regions per processor increases the execution time in most cases. This is because of the increased synchronization overhead that negates the benefits of the load distribution scheme. For 2 processors, having more Regions per processor leads to better load distribution and hence reduced execution time. The execution time settles around 14,000 seconds for 16 Regions or more. In this case too, contiguous cells are assigned to different processors and incur greater synchronization overhead for higher number of Regions per processor. Thus, the load balancing algorithm does not significantly impact the execution time for 16 processors and the trend can be expected to continue for higher number of processors. This indicates that the simulation load, represented by the number and type of events processed by a cell, remains relatively constant across the domain.

4.7 An Optimization

In the experiments described thus far each cell was mapped to an LP of the simulation engine. This provided maximum flexibility with respect to load balancing. However, this simplistic scheme has several drawbacks:

- It made every ion-transfer event go through multiple layers of the simulation engine, which caused needless event management overhead.

- There was little opportunity for an LP to execute multiple events within one context switch to that process. Redundant context switches were induced to dispatch events to cells that might have been adjacent to each other physically and hence communicated in a tightly coupled manner.
- Locality of communication was unexploited.
- The number of LPs in the simulation was equal to the number of cells. As a result, when the domain size was increased, the simulation engine had to synchronize and manage a large number of LPs. This was a barrier to scaling the domain to 3D scales.
- Shared state was disallowed in this scheme. Neighboring cells could not exchange data via direct access to data structures.

From previous experiments, it was found that all cells receive roughly an equal number of each event type (load) and that cells only communicate with their immediate neighbors, a better mapping of cells to LPs could be designed. The improved mapping is shown in Figure 11 [52].

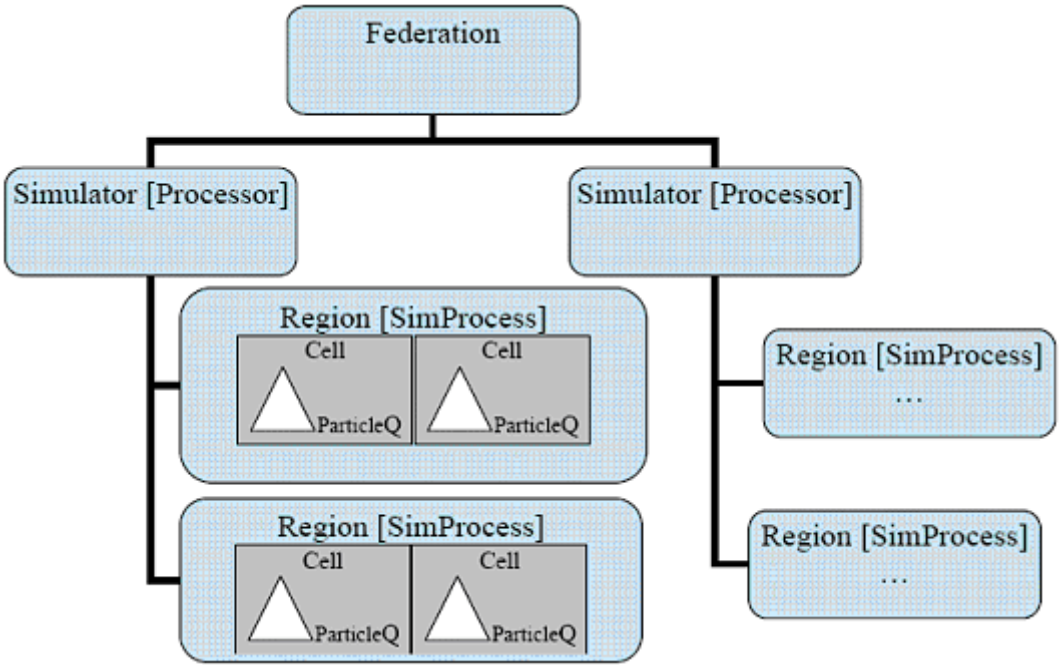

Figure 11 Improved Mapping of Cells to LPs

A "region" in this context is an aggregate that contains multiple cells. Instead of mapping one cell to an LP, a group of cells is mapped to each LP forming a region. The number of LPs that must be synchronized is reduced to a fraction of the number of cells. This method not only exploits locality of communication, but also allows the modeler to use shared-state across cells mapped to the same region.

To verify that this mapping was indeed more efficient, an experiment was carried out on a single processor with a domain size of 65,536 cells. Two simulations, one using the old mapping (65,536 LPs) and one with the new mapping (one LP) were compared. The simulation with one LP was more than twice as fast as the other. The same result was obtained even when the lookahead values of these models were varied. The scaling

properties of the simulation with the new method of mapping are expected to be superior. Research is currently under way in this area.

4.8 Performance Summary

Performance evaluation of the hybrid model revealed several areas of dependence between the modeling application and the simulation executive. Increasing the lookahead provides good parallel performance, while not greatly increasing the inaccuracy (less than 10% in certain scenarios). For larger number of processors, communication and synchronization overheads dominate the execution time for small number of cells. The simulation load, represented by the number and type of events processed by an LP, is equally distributed among all the LPs. We investigated an optimization that attempts to reduce the event management overhead of the simulation engine and exploits locality of communication. Extending the optimization to different scenarios across multiple processors is on-going research.

5. CONCLUSIONS

Advances in PDES research to date have had little impact in space physical science, where multi-physics and multi-scale physical systems are modeled by partial differential equations and particles. Traditionally, simplified models of such physical systems have been simulated using time-driven or time-stepped approaches. The inherent limitations of the time stepped approach prevents the simulation of more complex physical systems.

This thesis outlines a process for converting complex time-stepped codes in the scientific domain to their parallel discrete event counterparts, and applied to a hybrid plasma simulation model. Our work is designed to enable cross-disciplinary research by allowing scientists to develop simulation applications with little knowledge of underlying PDES mechanisms. By combining some of the machinery of time-stepped simulation (e.g., spatial grid generation, algorithms for solving the field equations) with the time advance method of event-driven simulations, an accurate parallel discrete event simulation is developed from a time-stepped simulation. Though overheads for synchronization and event exchange remain a challenge for large number of processors, the parallel discrete event model offers several advantages:

- *Built-in Irregular Time Stamps.* Unlike time-stepped simulations, relatively complex algorithms are avoided for achieving spatially varying time steps. The time advance is based on a queuing system approach and each event is processed based on its own temporal scale.
- *Intrinsic Algorithmic Intelligence.* Consider a disturbance launched from some point within the simulation domain. The disturbance can only propagate at a finite

velocity through a medium affecting a limited region around it. DES automatically takes advantage of this fact and concentrates the computations in the region of influence of the disturbance.

- *Superior Performance.* A sequential DES version of the hybrid plasma simulation model outperforms an equivalent fastest time-stepped model by a factor of over 100. Parallelization of the DES version potentially offers additional improvements in performance in proportion with the number of processors that are used. This in turn implies a potential speed up of up to10^4- 10^5 in three-dimensions. Such a performance increase will have a dramatic impact in many fields of science. To provide perspective, the Earth Simulator project promises to achieve a 1000-fold speedup of global climate studies using a combination of specialized software and hardware.
- *Less Restrictive CFL Condition.* Consider the simple problem of uniform plasma flowing at a finite velocity from left to right. In a stable code, this configuration maintains its properties, and density will remain constant. In an explicit electromagnetic full particle code, the criterion for stability is the speed-of-light CFL condition. Any larger speed can cause the code to become unstable although there are no light waves present in the system. Several schemes such as Darwin approximation and fully implicit schemes [41] have been devised to relax this condition to $dt < dx / v$ in such cases, where v is the particle velocity. These schemes come at the expense of considerable overhead and complexity and are generally hard to parallelize. In a discrete event simulation, however, it is relatively easier to address the problem. The stability condition in this case is that

no particle crosses more than one cell boundary in one exit time. As long as this condition is satisfied, the density of particles in each cell will remain the same and no event is triggered, thus assuring that the system remains stable.

- *Forecasting Capability*. The queuing system allows forecasts of the future evolution of the system. This facilitates the inclusion of sophisticated self-adjusting capability in the code (e.g., dynamically changing the cell size or the numerical model used in a given region).
- *Amenable to Multi-physics Problems*. Use of different physics models within the same simulation domain implies that different regions will evolve based on their own spatial and temporal scales. Discrete event methodology naturally decouples spatial and temporal scales and allows both to be distinct on a cell level.

The extension and evaluation of PDES systems in parallel computing environments containing thousands of processors is an on-going area of research. Several interesting avenues of research have emerged as a by-product of this thesis work. A key issue introduced by large-scale platforms is the increased delay of inter-processor communication. Using native implementations of communication libraries such as MPI on super-computing platforms could be a useful area of research. MPI implementations are optimized for performance on most large-scale clusters and compare favorably to TCP/IP based communication. MPI also provides synchronization primitives like global reductions and could be used by the parallel simulation executive as well as the application.

Another issue concerns the efficient realization of synchronization computations, such as determination of LBTS. The Null Message algorithm [6] exploits the static

communication topology between simulation processes. It has been shown to be efficient when simulation processes communicate with a small number of other simulation processes [7]. The hybrid model considered here exhibits this property. More sophisticated versions of the hybrid model are also likely to retain a fixed communication topology among simulation processes. The Null Message algorithm can be used in place of the Global Reduction algorithm, which synchronizes all simulation processes at each step, to reduce synchronization overheads. The layered architecture of Musik allows for using modular implementations of synchronization algorithms, without having to modify the application.

Scientific simulations may be considered to be valid within a certain statistical range (tolerance) of their output results. Therefore, it might be possible to introduce small causality errors into the system without violating vital physical properties. Using ideas from the Approximate Time-Stamps research [30] and the PEP algorithm [53], further improvements in performance could be obtained.

5. REFERENCES

1. Zeigler, B.P. et al. *Theory of Modeling and Simulation*, 2000: Academic Press.

2. Fujimoto, R.M., *Parallel and Distributed Simulation Systems*. 2000: Wiley Interscience

3. Bryant, R.E., *Simulation of packet communications architecture computer systems*, *MIT-LCS-TR-188*. 1977.

4. Peacock, J.K., J.W. Wong, and E.G. Manning, *Distributed Simulation Using a Network of Processors,* in *Computer Networks*, 1979.

5. Lamport, L., *Time, Clocks, and the Ordering of Events in a Distributed System,* in *Communications of the ACM*, 1978.

6. Chandy, K. and J. Misra, *Asynchronous distributed simulation via a sequence of parallel computations*, in *Communications of the ACM*. 1981.

7. Su, W.K. and C.L. Seitz, *Variants of the Chandy-Misra-Bryant Distributed Discrete Event Simulation Algorithm*, in *Proceedings of the SCS Multiconference on Distributed Simulation*. 1989.

8. Preiss, B., et al., *Null Message Cancellation in Conservative Distributed Simulation*, in *Advances in Parallel and Distributed Simulation*. 1991.

9. Groselj, B. and C. Tropper, *The Time of Next Event Algorithm*, in *Proceedings of the SCS Multiconference on Distributed Simulation*. 1988.

10. Cai, W. and S. Turner, *An Algorithm for Distributed Discrete-Event Simulation - The "Carrier Mull Message" Approach*, in *Distributed Simulation*: 1990, Society for Computer Simulation.

11. V. Sanchez et al*, Deblocking Event Algorithm: A New Approach to Conservative Parallel Discrete Event Simulation,* in *4th Euromicro Workshop on Parallel and Distributed Processing.*

12. Blanchard, T.D., et al, *Cooperative Acceleration: Robust Conservative Distributed Discrete Event Simulation*, in *Proceedings of the 1994 Workshop on Parallel and Distributed Simulation*. 1994.

13. Reynolds, P.F., Jr., *A Shared Resource Algorithm for Distributed Simulation*, in *Proceedings of the 9th Annual Symposium on Computer Architecture*. 1982.

14. Unger, B., et al., *Scheduling Critical Channels in Conservative Parallel Discrete Event Simulation*, in *Proceedings of the Workshop on Parallel and Distributed Simulation*. 1999.

15. Cota, B.A. and R.G. Sargent, *A Framework for Automatic Lookahead Computation in conservative Distributed Simulations*, in *Proceedings of the SCS Multiconference on Distributed Simulation*. 1990.

16. Nicol, D.M., *The Cost of Conservative Synchronization in Parallel Discrete Event Simulations,* in *Journal of the Association for Computing Machinery*, 1993.

17. Gan, B. P., et al., *Load balancing for conservative simulation on shared memory multiprocessor systems*, in *Workshop on Parallel and Distributed Simulation*, 2000.

18. Jefferson, D. R., *Virtual Time*, in *ACM Transactions on Programming Languages and Systems*.

19. Sokol, L.M. and B.K. Stucky, *MTW: Experimental Results for a Constrained Optimistic Scheduling Paradigm*, in *Proceedings of the SCS Multiconference on Distributed Simulation*. 1990.

20. Damani, O.P., Y.-M. Wang, and V.K. Garg, *Optimistic Distributed Simulation Based on Transitive Dependency Tracking*, in *Proceedings of the 11th Workshop on Parallel and Distributed Simulation*. 1997.

21. Prakash, A. and R. Subramanian, *An Efficient Optimistic Distributed Scheme Based on Conditional Knowledge*, in *Proceedings of the 6th Workshop on Parallel and Distributed Simulation*. 1992.

22. Turner, S. and M. Xu, *Performance Evaluation of the Bounded Time Warp Algorithm*, in *Proceedings of the 6th Workshop on Parallel and Distributed Simulation*. 1992.

23. Fujimoto, R.M., *Time Warp on a Shared Memory Multiprocessor.* Transactions of the Society for Computer Simulation, 1989.

24. Cleary, J., et al., *Cost of State Saving and Rollback*, in *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*. 1994.

25. Tay, S.C., Y.M. Teo, and S.T. Kong, *Speculative Parallel Simulation with an Adaptive Throttle Scheme*, in *Proceedings of the 11th Workshop on Parallel and Distributed Simulation*. 1997.

26. Bagrodia, R., K.M. Chandy, and W.T. Liao, *A Unifying Framework for Distributed Simulation. ACM Transactions on Modeling and Computer Simulation*, 1991.

27. Steinman, J., *SPEEDES: Synchronous Parallel Environment for Emulation and Discrete Event Simulation*, in *Advances in Parallel and Distributed Simulation*. 1991.

28. Perumalla, K.S., *_sik - A Micro-kernel for Parallel/Distributed Simulation,* in *ACM/IEEE/SCS Workshop on Parallel and Distributed Simulation.* 2005.

29. Carothers, C. et al, *Efficient Optimistic Parallel Simulations using Reverse Computation,* in *ACM/IEEE/SCS Workshop on Parallel and Distributed Simulation.* 1999.

30. Allen, J.F., *Maintaining Knowledge about Temporal Intervals,* in *Communications of the ACM*, 1983.

31. Diehl, C. and C. Jard. *Interval Approximations and Message Causality in Distributed Systems*, in *Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science.* 1992.

32. Dubois, D. and H. Prade, *Processing Fuzzy Temporal Knowledge,* in *IEEE Transactions on Systems, Man, and Cybernetics*, 1989.

33. Loper, M. and Fujimoto, R. *Exploiting Temporal Uncertainty in Process-Oriented Distributed Simulation*, in *Proceedings of Winter Simulation Conference*, 2004.

34. Loper, M. and R. Fujimoto, *Pre-Sampling as an Approach for Exploiting Temporal Uncertainty*, in *Proceedings of the 14th Workshop on Parallel and Distributed Simulation*. 2000.

35. Rajaei, H., R. Ayani, and L.-E. Thorelli, *The Local Time Warp Approach to Parallel Simulation*, in *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*. 1993.

36. Rao, D. M., N. V. Thondugulam, et al, *Unsynchronized Parallel Discrete Event Simulation*, in *Proceedings of the Winter Simulation Conference*, 1998.

37. Fujimoto, *Exploiting temporal uncertainty in parallel and distributed simulations,* in *Workshop on Parallel and Distributed Simulation,* 1999.

38. Taylor, T., *Class Drafts of "N-Body Problem",* in *Algorithms in the Real World, CMU,* 1999.

39. Makino, J. et al, *GRAPE-4: A Special-Purpose Computer for Gravitational N-body Problems,* in *Proceedings of Supercomputing*, 1995.

40. Hines, *A* Birdsall, C.K. and A.B. Langdon, *Plasma Physics via Computer Simulation*. 1985: McGraw-Hill Book Company.

41. Winske, D., et al. *Hybrid Codes: Past, Present and Future*, in *ISSS-6 Tutorials*. 2002. Springer-Verlag.

42. Karimabadi, H. and N. Omidi. *Latest Advances in Hybrid Codes and their Application to Global Magnetospheric Simulations*. in *GEM,* 2002.

43. Karimabadi, H, Driscoll, J, Omelchenko, Y.A. and Omidi N., *A New Asynchronous Methodology for Modeling of Physical Systems: Breaking the Curse of Courant Condition*, in *Journal of Computational Physics,* 2005.

44. Winske, D. and N. Omidi, *Hybrid codes: Methods and Applications*, in *Computer Space Plasma Physics: Simulation Techniques and Software*. 1993. Terra Scientific Publishing Company.

45. Karimabadi, H., Omidi N., and Kazeminezhad F., *Global Hybrid Simulations: A Progress Report*, in *Eos Transactions AGU*, 2002.

46. Omidi, N., et al., *Hybrid Simulations of Solar Wind Interaction with Magnetized Asteroids: General Characteristics,* in *Journal of Geophysics Research*, 2002.

47. Dawson, C. and R. Kirby, *High Resolution Schemes for Conservation Laws with Locally Varying Time Steps,* in *SIAM Journal of Scientific Computing*, 2001.

48. Gibbons, M.R. and D.W. Hewett, *The Darwin Direct Implicit Particle-in-cell (DADIPIC) Method for Simulation of Low Frequency Plasma Phenomena,* in *Journal of Computational Physics*, 1995.

49. Neilson, C.W. and E.L. Lindman. *An Implicit Two-dimensional Electromagnetic Plasma Simulation Code*, in *Sixth Conference on Numerical Simulation of Plasmas*. 1973.

50. Vu, H.X. and J.U. Brackbill, *CELESTID: An Implicit Fully-Kinetic Model for Low-Frequency Electromagnetic Plasma Simulation,* in *Computational Physics Communications*, 1992.

51. Perumalla K. and Fujimoto R.M., *Virtual Time Synchronization over Unreliable Network Transport,* in *ACM/IEEE/SCS Workshop on Parallel and Distributed Simulation,* 2001.

52. Perumalla K., *Musik Performance Report*, 2004.

53. Yuri Omelchenko, *SciDES- Tools for Scientific Discrete Event Simulation*, Scibernet Inc. 2004.